

@title 信息论第一次大作业

@author 18374480-黄翔

实验环境

- python 2.7.18
 - 库：os, sys, argparse, math
-

必选模组——模组2

功能实现

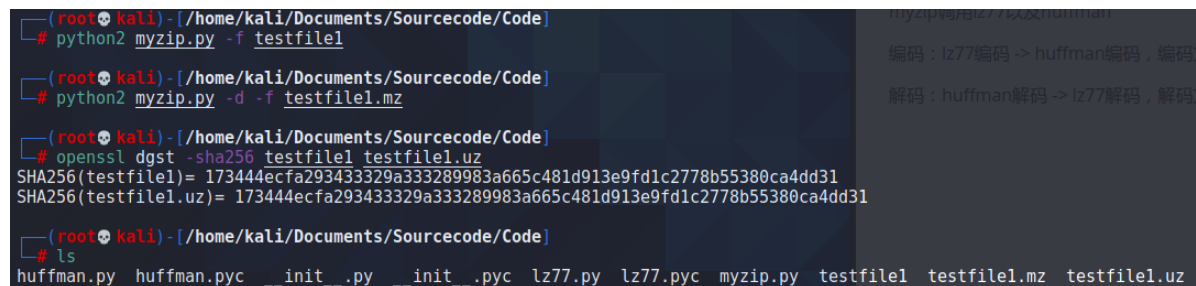
myzip调用lz77以及huffman

编码：lz77编码 -> huffman编码，编码文件后缀为.mz

解码：huffman解码 -> lz77解码，解码文件后缀.mz改为.uz

对于huffman编码，额外信息写入了magic number，最后字节0填充个数，码表长度，码表，编码结果

对于LZ77编码，当在滑动窗口没有找到匹配时，写入(0, char(8 bits))，char为前向缓冲区最靠近滑动窗口的那个字符；当找到匹配时，写入(1, 距离(12 bits)，匹配长度(4 bits))



```
(root@kali) - [/home/kali/Documents/Sourcecode/Code]
# python2 myzip.py -f testfile1

(root@kali) - [/home/kali/Documents/Sourcecode/Code]
# python2 myzip.py -d -f testfile1.mz

(root@kali) - [/home/kali/Documents/Sourcecode/Code]
# openssl dgst -sha256 testfile1 testfile1.uz
SHA256(testfile1)= 173444ecfa293433329a333289983a665c481d913e9fd1c2778b55380ca4dd31
SHA256(testfile1.uz)= 173444ecfa293433329a333289983a665c481d913e9fd1c2778b55380ca4dd31

(root@kali) - [/home/kali/Documents/Sourcecode/Code]
# ls
huffman.py  huffman.pyc  __init__.py  __init__.pyc  lz77.py  lz77.pyc  myzip.py  testfile1  testfile1.mz  testfile1.uz
```

用户交互

使用python argparse库设计

```

└─# python2 myzip.py --help
usage: myzip [OPTION]... [FILE]...

Compress or uncompress files using huffman and LZ77
the compress file will end with .mz, and the uncompress file will end with .uz
Example:
cat file | python2 myzip.py
cat file.mz | python2 myzip.py -d
python2 myzip.py file1 file2
python2 myzip.py -d file1.mz file2.mz

optional arguments:
-h, --help            show this help message and exit
-f [FILE [FILE ...]], --file [FILE [FILE ...]]
                        target file
-d, --decompress      uncompress the file

With no FILE, or when FILE is -, read standard input.

```

代码鲁棒性与安全性

- 使用Magic Number在文件头标记文件：huffman模块为编码最后一步与解码第一步，通过huffman模块加上magic number，对于文件模式，magic number为0xff; 对于标准输入输出，由于使用缓冲区，在每次缓冲区满编码时加上magic number: 0xfe
- 通过判断后缀.mz，.uz在文件名层面标记编码、解码文件

可选模组

分别对huffman模块与lz77模块实现编码解码对标准输入输出的支持

缓冲区设置为1000 bytes，缓冲区满后，进行编码。结束判断缓冲区是否为空，不为空则也作为一个块编码输出

对于myzip模块，使用管道将huffman模块与lz77模块相连。当没有目标文件或者目标文件名为“-”时使用标准输入输出

```

if filename == '-': #pipe
    r,w = os.pipe()
    pid = os.fork()
    if pid: # parent process
        os.wait()
        os.close(w)
        saved_stdin = sys.stdin
        sys.stdin = os.fdopen(r, 'rb') #redirect the stdin to pipe in for parent
        LZ77().decode2()
        sys.stdin.close()
        sys.stdin = saved_stdin
    else: #child process
        os.close(r)
        saved_stdout = sys.stdout
        sys.stdout = os.fdopen(w, 'wb') #redirect the stdout to pipe out for child
        huffman.decode2()
        sys.stdout.close()
        sys.stdout = saved_stdout
        sys.exit(0)
    break

```

代码鲁棒性与安

- 使用Magic
- huffman模
- 使用缓冲区
- 通过判断后

可选模组

分别对huffmar

对于myzip模块

直接输出到标准输出的展示

```
└─# cat testfile1 | python2 myzip.py
00x001
00? Y0000 00( 00A408 0<{0m00
0Hn0 0PB0 0X
00;|0010 0p 00 0Cx|"000 00
0
0D
W0 0E/)0C02F0oKX0[ L00
0k 0G000H0< 0p30
0
0qY00#0
00$
00
00>Z0s00I00
N00%)BJ 0-j0
0= Y0r0^0K0 0iX0%{00000000y0000Ds
00 0000a000-00 00007ll+0=Y000
?n00i0000T^0,U0000!(0;1Vo00N0090=000yN!300/0C0_00 $00000P0>01000a]{00^=006U00000000`0060000Ws[0?000]Q0v0000
00%000>00_0000
0eYh00=.0000`0060000W 0?00%[]Q0v00g}00Y000tG<00b<000000m0<00E00W
00%0000000T0(00aC 0X000000W000M?0S0SIV070y,'Fxp000
0t0"0X0T0S000/0:-000-000v003#~0000z00K0?.0ZL000070.000000000,j0Sm070s'0000P0>01000aQ"0[0r0(0@0X0000)00|00000M$0,0)0
0
F/*0+00Y G0`1,
00 ^240 08
0<10000"D
00
0HiPt300`~0 00H0 040 00
0x|500u0
0
```

为了说明程序正确性，在终端将输出结果重定向到文件

```
└─(rootkali) - [/home/kali/Documents/Sourcecode/Code]
└─# cat testfile1 | python2 myzip.py > testfile2.mz

└─(rootkali) - [/home/kali/Documents/Sourcecode/Code]
└─# cat testfile2.mz | python2 myzip.py -d > testfile2.uz

└─(rootkali) - [/home/kali/Documents/Sourcecode/Code]
└─# openssl dgst -sha256 testfile1 testfile2.uz
SHA256(testfile1)= 173444ecfa293433329a333289983a665c481d913e9fd1c2778b55380ca4dd31
SHA256(testfile2.uz)= 173444ecfa293433329a333289983a665c481d913e9fd1c2778b55380ca4dd31
```

实践问题

重复性的文件结构

直接进行huffman编码会发现码表中每个字节(8 bits)对应的编码也为 8 bits，（加上必要信息如码树后编码文件反而更大。

使用lz77编码再使用huffman编码，lz77编码对重复性的文件内容压缩效果非常好（当然，这与选择的滑动窗口大小有关），lz77编码破坏了文件重复性结构，之后进行huffman编码就能避免直接用huffman编码的问题，得以进一步压缩文件

```
-rw-r--r-- 1 root root 65536 Apr 17 15:50 testfile1
-rw-r--r-- 1 root root 9786 Apr 25 11:51 testfile1.mz
-rw-r--r-- 1 root root 65536 Apr 25 11:51 testfile1.uz
```

不同格式的压缩

jpeg格式：

```
-rw-r--r-- 1 root root 113040 Apr 25 13:20 picture.jpg
-rw-r--r-- 1 root root 201256 Apr 25 13:20 picture.jpg.mz
-rw-r--r-- 1 root root 113040 Apr 25 13:22 picture.jpg.uz
```

bmp格式：

```
-rw-r--r-- 1 kali kali 2764938 Apr 25 13:26 picture.bmp
-rw-r--r-- 1 root root 1832141 Apr 25 14:00 picture.bmp.mz
```

jpeg格式已经是经过很好压缩之后的格式，编码时很难再次减少其体积（反而因为自己编码方法的存储优化不足等导致文件更大。。）

BMP（全称Bitmap）是Windows操作系统中的标准图像文件格式，可以分成两类：设备相关位图（DDB）和设备无关位图（DIB），使用非常广。它采用位映射存储格式，除了图像深度可选以外，不采用其他任何压缩，因此，BMP文件所占用的空间很大。对其编码压缩效果比较明显。

可执行文件：

直接使用实验附带的generate_testfile.c生成的可执行文件

```
-rwxr-xr-x 1 root root 19872 Apr 27 18:07 generate_testfile
-rw-r--r-- 1 kali kali 406 Apr 8 21:12 generate_testfile.c
-rw-r--r-- 1 root root 7716 Apr 27 18:08 generate_testfile.mz
```

可见，文件被压缩到原文件一半大小左右

查看原可执行文件，可以看到文件中存在大量的0字节，因此对其进行lz77编码效果以及huffman编码效果很好

```
00004c70: 0000 0000 0000 0000 ed36 0000 0000 0000 .....6.....
00004c80: 7c02 0000 0000 0000 0000 0000 0000 0000 .....
00004c90: 0100 0000 0000 0000 0100 0000 0000 0000 .....
00004ca0: 4701 0000 0100 0000 0000 0000 0000 0000 G.....
00004cb0: 0000 0000 0000 0000 6939 0000 0000 0000 .....i9.....
00004cc0: c300 0000 0000 0000 0000 0000 0000 0000 .....
00004cd0: 0100 0000 0000 0000 0000 0000 0000 0000 .....
00004ce0: 0100 0000 0200 0000 0000 0000 0000 0000 .....
00004cf0: 0000 0000 0000 0000 303a 0000 0000 0000 .....0:.....
00004d00: d806 0000 0000 0000 2200 0000 3300 0000 ..... " ...3...
00004d10: 0800 0000 0000 0000 1800 0000 0000 0000 .....
00004d20: 0900 0000 0300 0000 0000 0000 0000 0000 .....
00004d30: 0000 0000 0000 0000 0841 0000 0000 0000 .....A.....
00004d40: 4302 0000 0000 0000 0000 0000 0000 0000 .....C.....
00004d50: 0100 0000 0000 0000 0000 0000 0000 0000 .....
00004d60: 1100 0000 0300 0000 0000 0000 0000 0000 .....
00004d70: 0000 0000 0000 0000 4b43 0000 0000 0000 .....KC.....
00004d80: 5201 0000 0000 0000 0000 0000 0000 0000 .....R.....
00004d90: 0100 0000 0000 0000 0000 0000 0000 0000 .....

```

黑洞

```
-rw-r--r-- 1 kali kali 2764938 Apr 25 13:26 picture.bmp
-rw-r--r-- 1 root root 1832141 Apr 25 14:00 picture.bmp.mz
-rw-r--r-- 1 root root 3291973 Apr 25 19:42 picture.bmp.mz.mz
```

无法无限压缩。由香农第一定理， $\frac{L_n}{n}$ 最多无限趋近于信源熵 $H_r(S)$ 。

换个角度说，如果最终所有信息都可以压缩到1KB，每个压缩结果必定会表示多个可能的信息，因此无法恢复出原信息，不是无损压缩。

